

Technical Report
769

A Comparison of Hamming and Hopfield Neural Nets for Pattern Classification

R.P. Lippmann
B. Gold
M.L. Malpass

21 May 1987

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS



Prepared for the Department of the Air Force
under Electronic Systems Division Contract F19628-85-C-0002.

Approved for public release; distribution unlimited.

ADA 182255

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the support of the Department of the Air Force under Contract F19628-85-C-0002.

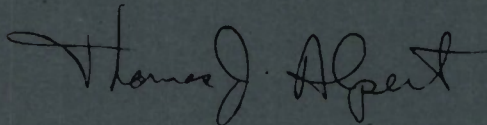
This report may be reproduced to satisfy needs of U.S. Government agencies.

The views and conclusions contained in this document are those of the contractor and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the United States Government.

The ESD Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

A handwritten signature in dark ink, reading "Thomas J. Alpert". The signature is fluid and cursive, with the first name "Thomas" and last name "Alpert" clearly legible.

Thomas J. Alpert, Major, USAF
Chief, ESD Lincoln Laboratory Project Office

Non-Lincoln Recipients

PLEASE DO NOT RETURN

Permission is given to destroy this document
when it is no longer needed.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

**A COMPARISON OF HAMMING AND HOPFIELD
NEURAL NETS FOR PATTERN CLASSIFICATION**

*R.P. LIPPMANN
B. GOLD
M.L. MALPASS
Group 24*

TECHNICAL REPORT 769

21 MAY 1987

Approved for public release; distribution unlimited.

LEXINGTON

MASSACHUSETTS

ABSTRACT

The Hopfield model neural net has attracted much recent attention. One use of the Hopfield net is as a highly parallel content-addressable memory, where retrieval is possible although the input is corrupted by noise. For binary input patterns, an alternate approach is to compute Hamming distances between the input pattern and each of the stored patterns and retrieve that stored pattern with minimum Hamming distance. We first show that this is an optimum processor when the noise is statistically independent from bit to bit. We then present a Hamming Neural Net which is a highly parallel implementation of this algorithm that uses computational elements similar to those used in a Hopfield net. We also compare the Hopfield and Hamming nets for several applications. For the cases considered, the Hamming net generally outperforms the Hopfield net. Also, the Hamming net requires fewer interconnects than the fully connected Hopfield net.

TABLE OF CONTENTS

Abstract	iii
List of Illustrations	vii
List of Tables	ix
1. Introduction	1
2. Optimum Processor for Classification of Binary Patterns	3
3. Neural Net Implementation of Optimum Processor	7
4. The Hopfield Classifier	21
5. Comparisons between Hopfield Classifiers and the Hamming Net	27
6. Conclusions	35
References	37

LIST OF ILLUSTRATIONS

Figure No.		Page
1	Generation of Corrupted Bit Pattern by Passing the Exemplar for Pattern Class j through a Noisy Discrete Memoryless Channel.	3
2	Block Diagram of Optimum Maximum Likelihood Classifier.	4
3	Feed-Forward Perceptron Neural Net Used to Calculate Values Related to the Likelihood of Each of M Pattern Classes for Patterns with N Elements. All Nodes Are Analog-Threshold Logic Nodes with Internal Thresholds Set to Zero. Weights Depend on the Stored States.	8
4	Feed-Forward Neural Net That Determines Which of Eight Inputs Is Maximum by Explicitly Performing All Binary Comparisons. Nodes Denoted by Filled Circles Are Hard-Limit Nodes and Nodes Denoted by Open Circles Are Analog-Threshold Logic Nodes. Internal Thresholds on All Nodes except the Output Nodes Are Zero. Internal Thresholds on Nodes $z_0, z_1, \dots, z_6, z_7$ Are $-6.5, -5.5, \dots, -0.5, +0.5$, Respectively. Weights Are Either $+1$ (Open Arrows) Or -1 (Filled Arrows).	10
5	Comparator Subnet That Selects the Maximum of Two Inputs. Internal Thresholds on both Hard-Limit Nodes (Filled Circles) and Analog-Threshold Logic Nodes (Open Circles) Are Zero.	11
6	Feed-Forward Neural Net That Determines Which of Eight Inputs Is Maximum Using a Binary Tree and Comparator Subnets from Figure 5. Internal Thresholds on Both Hard-Limit Nodes (Filled Circles) and Analog-Threshold Logic Nodes (Open Circles) Are Zero Except for the Output Nodes. Internal Thresholds on Nodes $z_0, z_1, \dots, z_6, z_7$ Are -2.5 . Weights For All Comparator Subnets in This Net Are As in Figure 5. All Other Weights Are $+1$.	11
7	Iterative Neural Net Denoted "Maxnet" That Determines Which of M Inputs Is the Maximum. The Inputs Are Applied Prior to Time Zero and Then Removed, and the Outputs Are Valid after the Network Converges. All Nodes Are Analog-Threshold Logic Nodes with Internal Thresholds Equal to Zero. Each Node Connects to Itself and All Other Nodes. Weights Are $-w$ (Solid Arrows) Or $+1$ (Open Arrows).	12

Figure No.		Page
8	Node Outputs in a 10-Node Maxnet at Iterations Zero through 3 When the Net Converged. Initial Values Come from the Perceptron Net Presented in Figure 3 When the Number of Classes (M) Is 10, the Number of Input Nodes (N) Is 100, and the Input to the Perceptron Net Is the Exemplar Pattern for Class Number 5.	14
9	Node Outputs in a 100-Node Maxnet at Iterations Zero through 9 When the Net Converged. Initial Values Come from the Perceptron Net Presented in Figure 3 When the Number of Classes (M) Is 100, the Number of Input Nodes (N) Is 1000, and the Input to the Classifier Is the Exemplar Pattern for Class Number 50.	16
10	Node Outputs in the 100-Node Maxnet of Figure 9 When the Input to the Classifier Is the Exemplar Pattern for Class Number 50 after Being Passed through a Memoryless Binary-Symmetric Channel with an Error Probability, ρ , of 0.4.	17
11	Average Number of Iterations until Convergence for the Maxnet of Figure 7. Initial Values Come from the Perceptron Net Presented in Figure 3 When the Number of Classes (M) Ranges from 2 to 10, the Number of Input Nodes (N) Is 10M, and the Input Is the Exemplar for One Pattern after Being Passed through a Binary-Symmetric Channel with the Specified Probability of Error.	18
12	Complete Optimum Neural-Net Classifier Referred to as a Hamming Net Made Up of a Perceptron to Calculate Likelihoods, and a Maxnet to Select the Node with the Maximum Likelihood.	19
13	Iterative Hopfield Neural Net. The Inputs Are Applied Prior to Time Zero and Then Removed, and the Outputs Are Valid after the Network Converges. Nodes Denoted by Filled Circles Are Symmetric Hard-Limit Nodes, and Nodes Denoted by Open Circles Are Analog-Threshold Logic Nodes. Internal Thresholds in All Nodes Are Set to Zero. Each Node in the Middle Row Connects to All Other Nodes but Not to Itself. Weights Are Specified by t_{ij} (Solid Arrows) or Are +1 (Open Arrows).	22
14	Complete Hopfield Neural-Net Classifier Made Up from a Hopfield Net Followed by a Perceptron. The Hopfield Net Is as in Figure 13. The Perceptron Is Designed as in Figure 3 except All Nodes Are Hard-Limit Nodes and the Internal Thresholds in the Final Output Nodes Are Set to $\epsilon - N$ Instead of Zero, Where $\epsilon < 1$.	24

Figure No.		Page
15	Digit Patterns Used in Experiment No. 1.	27
16	Results of Experiment No. 1 Obtained Using Digit Patterns (a) and Random Patterns (b) and a Hopfield Net without Orthogonalization.	29
17	Hopfield Net (a) and Hopfield-Hamming Net (b) Performance with and without Orthogonalization Using Digit Patterns from Experiment No. 1.	32
18	Hopfield Net (a) and Hopfield-Hamming Net (b) Performance with Orthogonalization Using 16 Hexadecimal Digits and Using 8 Digit Patterns as Exemplars.	33

LIST OF TABLES

Table No.		Page
I	Comparison of Three Neural Nets That Pick the Maximum of M Inputs. (Node Counts Do Not Include Input or Output Nodes.)	19
II	Comparison of Three Different Neural Nets That Can Be Used to Classify M Binary Patterns When Each Pattern Has N Elements. (Counts of Numbers of Nodes and Interconnects Do Not Include Input and Output Nodes or Interconnects Between Major Internal Subnets.)	25
III	Hamming Distances between Digit Patterns Used in Experiment No. 1.	28
IV	Hamming Distances between Random Patterns Used in Experiment No. 2.	28
V	Exemplar Bibliography Patterns Used in Experiment No. 3.	30

A COMPARISON OF HAMMING AND HOPFIELD NEURAL NETS FOR PATTERN CLASSIFICATION

1. INTRODUCTION

There has been a recent upsurge of interest in neural net models made of highly parallel computational elements connected in a pattern that is reminiscent of biological neural nets. In particular, much recent work has explored the ability of a neural model described by Hopfield^{1, 3} to serve as a content-addressable memory and as a pattern classifier for binary bit patterns. A content-addressable memory retrieves one of M stored patterns given an input pattern which is a noisy version of a stored pattern. A classifier determines which of M exemplar patterns is most similar to a noisy input pattern. In the following we focus on the classification problem because a content-addressable memory is essentially a classifier which outputs the exemplar for the selected class instead of an index to the class. We also focus on the classification problem because classification is a fundamental operation that is essential to the important problems of speech and image recognition whether achieved by biological or artificial means.

Past studies have demonstrated that the Hopfield model can be used as a content-addressable memory for random input patterns^{1, 3} and to classify binary patterns created from radar cross sections⁴, from consonants and vowels extracted from spoken words⁵, and from lines in an image⁶. These results demonstrate that a neural net based on the Hopfield model can perform classification. In addition, Hopfield models have been successfully applied to other problems, such as the travelling salesman problem, the A-D converter problem, and the signal decomposition problem^{7, 8}.

We have been interested in a specific set of pattern classification problems in speech and image processing. In some special cases, such problems can be formulated in maximum-likelihood terms and optimum processors can be derived. In particular, if each element in a binary pattern is perturbed independently by noise, the optimum processor is an algorithm that measures Hamming distances between the perturbed input pattern and each of the stored patterns, and selects the minimum.

In this report, we derive this optimum-processor result and then show how a neural net model called the Hamming net can be constructed to perform this algorithm. We then compare implementations and performance of the Hopfield and Hamming nets using simulations of a visual digit recognition task and a bibliography retrieval task.

2. OPTIMUM PROCESSOR FOR CLASSIFICATION OF BINARY PATTERNS

An optimum binary classifier must classify each binary input vector \mathbf{x} into one of M classes such that the probability of a classification error is minimized. Here, the input vector \mathbf{x} has N elements which can be in one of two states denoted the $+1$ and the -1 states. Each of the M classes is represented by an exemplar binary vector \mathbf{x}^j where $j = 0, 1, 2, \dots, M-1$ is the index to the class.

The classifier can be easily analyzed if each input vector to be classified is obtained by passing each component of an exemplar through a discrete, noisy, memoryless channel, as shown in Figure 1; i.e., the noise is independent from bit to bit. In Figure 1 the value of an element in the $+1$ and -1 state is taken to be $+1$ and -1 , respectively. In some further formulations these values will be $+1$ and 0 instead.



Figure 1. Generation of corrupted bit pattern by passing the exemplar for pattern class j through a noisy discrete memoryless channel.

The channel in Figure 1 is defined by four conditional probabilities:

$$\begin{aligned}
 P(-1|+1) &= \epsilon \\
 P(+1|+1) &= 1 - \epsilon \\
 P(+1|-1) &= \rho \\
 P(-1|-1) &= 1 - \rho
 \end{aligned} \tag{1}$$

where

$$0.0 \leq \epsilon < 0.5 \text{ and } 0.0 \leq \rho < 0.5 \tag{2}$$

Each probability in Equation (1) is the conditional probability of observing a specific bit at the output of the channel given a specific bit at the input.

The binary classification problem created by a discrete memoryless channel is identical to the classical communication theory problem of building a decoder to determine which of M block codes of length N was sent over a noisy discrete memoryless channel. In our terminology, however, the block codes are the exemplars. If the *a priori* probabilities of presenting exemplars from

different classes at the input to the noisy channel are equal, then the minimum error decoder is a maximum likelihood decoder⁹ that selects the class j^* for which

$$P(\mathbf{x}|\mathbf{x}^{j^*}) \geq P(\mathbf{x}|\mathbf{x}^j); \text{ all } j \neq j^* \quad (3)$$

In this equation, $P(\mathbf{x}|\mathbf{x}^j)$ is the likelihood for class j or the conditional probability of observing the vector \mathbf{x} at the output of the noisy channel, given that exemplar \mathbf{x}^j was presented at the input. A block diagram of an optimum maximum likelihood classifier is presented in Figure 2. In this figure the input vector \mathbf{x} is presented at the left, likelihood values are calculated in parallel, and then the class with the maximum likelihood value is selected. Likelihood values are denoted y_j , where $y_j = p(\mathbf{x}|\mathbf{x}^j)$. The output is a vector \mathbf{z} whose elements are zero except for that element corresponding to the class j^* that satisfies Equation (3).

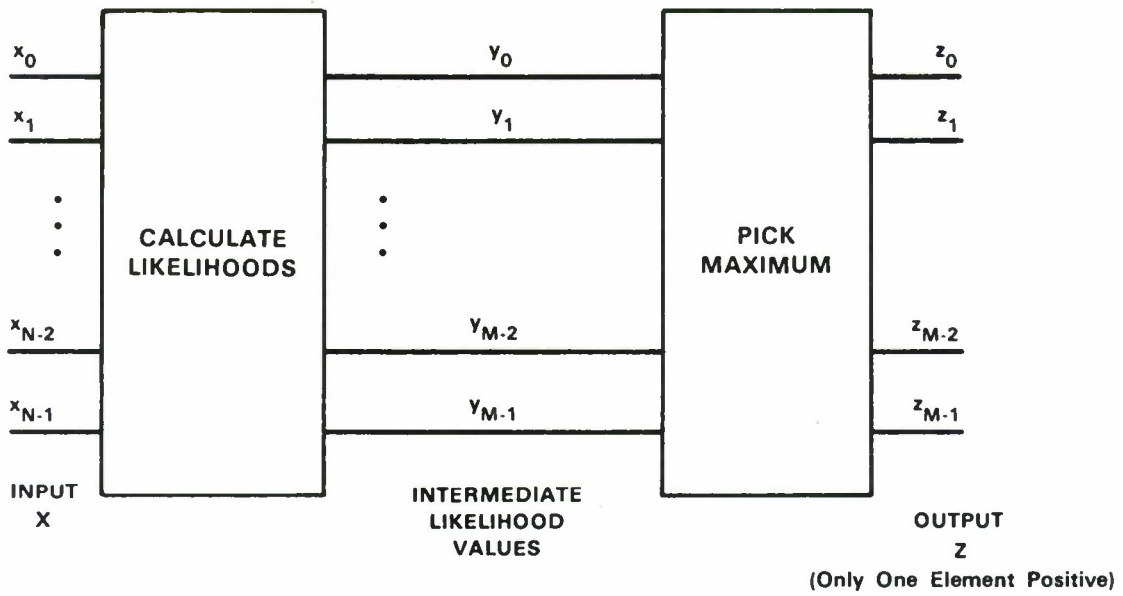


Figure 2. Block diagram of optimum maximum likelihood classifier.

The exact form of the maximum likelihood decoder depends on the probabilities that define the noisy channel. In all cases, however, we will show that the likelihoods are monotonically related to functions equal to weighted sums of elements from the input vector. For example, in the simplest case a binary symmetric channel is used and $\rho = \epsilon$. In this case it is equally likely for a +1 state to be changed to a -1 state and vice versa and

$$P(\mathbf{x}|\mathbf{x}^j) = \epsilon^{N_{\text{ham}}^j} (1 - \epsilon)^{N - N_{\text{ham}}^j} \quad (4a)$$

where N_{ham}^j is the Hamming distance between x and x^j . This is the number of elements in the input which are not identical to the corresponding element in the exemplar for class j . Equation (4a) simplifies to

$$P(x|x^j) = \left(\frac{\epsilon}{1-\epsilon} \right)^{N_{\text{ham}}^j} (1-\epsilon)^N \quad (4b)$$

Since ϵ is less than 0.5, the first fraction on the right is less than 1.0 and Equation (4b) is a maximum for that stored state with the smallest Hamming distance to the input. A neural net that implements an optimum classifier for the binary symmetric channel must thus calculate the Hamming distance to exemplars for all classes and then select that class which produces a minimum. Instead of calculating the Hamming distance directly, we will calculate N minus the Hamming distance and maximize this function.

N minus the Hamming distance can be calculated from a weighted sum of the elements of the input vector. If the elements of the input vector take on the values $+1$ and -1 for the $+1$ and -1 states, respectively, then

$$N - N_{\text{ham}}^j = c_j + \sum_{i=0}^{N-1} w_{ij} x_i \quad (5a)$$

In this equation,

$$w_{ij} = \frac{x_i^j}{2} \quad (5b)$$

and

$$c_j = \frac{N}{2} \quad (5c)$$

Here x_i^j is the value of element i of the exemplar for class j . When all elements in the input vector match an exemplar exactly, each element in the sum of Equation (5a) adds $1/2$, and the total is N . Whenever an element in the input vector doesn't match the corresponding element in the exemplar, the prior total is decremented by 1 as required.

An alternative derivation was suggested in¹¹ where it is assumed that elements of the input vector x take on the values 0 and $+1$ for the -1 and $+1$ states, respectively. In this case N minus the Hamming distance can be calculated from:

$$N - N_{\text{ham}}^j = c_j + \sum_{i=0}^{N-1} w_{ij} x_i \quad (6a)$$

In this equation,

$$w_{ij} = \begin{cases} +1 & \text{if } x_i^j = +1 \\ -1 & \text{if } x_i^j = 0 \end{cases}, \quad (6b)$$

$$c_j = N_z^j = N - \sum_{i=0}^{N-1} x_i^j \quad (6c)$$

Here N_z^j represents the number of elements in the exemplar for class j that are zero. When all elements in the input vector match an exemplar exactly, the sum in (6a) adds up the number of positive input elements. This, added to the number of zero input elements results in N as desired. The sum is reduced by one whenever a zero input element that matches an exemplar becomes positive, and whenever a positive input element that matches an exemplar becomes zero.

In the more general situation, the noisy channel is not symmetric and $\rho \neq \epsilon$. In this case it is more likely that the $+1$ state will change to the -1 state or vice versa, and it is not sufficient to simply calculate the Hamming distance. For simplicity, we present results for the case when elements of the input vector \mathbf{x} take on the values 0 and $+1$ for the -1 and $+1$ states, respectively. We will also maximize $\log P(\mathbf{x}|\mathbf{x}^j)$, denoted L_j , instead of $P(\mathbf{x}|\mathbf{x}^j)$. In this case

$$L_j = \sum_{\forall x_i^j = 1} x_i \log \left(\frac{1-\epsilon}{\epsilon} \right) + (N - N_z^j) \log(\epsilon) + \sum_{\forall x_i^j = 0} x_i \log \left(\frac{\rho}{1-\rho} \right) + N_z^j \log(1-\rho) \quad (7a)$$

This can be written as a weighted sum of elements of the input vector as was done in Equation (5a) and Equation (6a):

$$L_j = c_j + \sum_{i=0}^{N-1} w_{ij} x_i \quad (7b)$$

In this equation

$$w_{ij} = \begin{cases} \log \left(\frac{1-\epsilon}{\epsilon} \right) & \text{if } x_i^j = +1 \\ \log \left(\frac{\rho}{1-\rho} \right) & \text{if } x_i^j = 0 \end{cases}, \quad (7c)$$

$$c_j = (N - N_z^j) \log(\epsilon) + N_z^j \log(1-\rho), \quad (7d)$$

with N_z^j is as in (6c). When $\rho = \epsilon$, (7a) reduces to a form similar to Equation (6a).

3. NEURAL NET IMPLEMENTATION OF OPTIMUM PROCESSOR

In the previous section it was demonstrated that the optimum processor always forms weighted sums of the elements of the input vector and picks the maximum from these sums. In this section we demonstrate that this processor can be implemented using an artificial neural net. The question of what algorithms can be implemented using neural nets is of interest because of the potential usefulness of such nets.

An artificial neural net is a highly parallel network with many interconnections between analog computational elements or nodes. The simplest node forms the sum of N weighted inputs presented on N input links and passes the result through a nonlinearity out on one output link. Neural nets almost always include an inherent nonlinearity and require primarily local connectivity between nodes. In addition, the weights on the input links can be adapted based on information concerning the correctness of the output. Artificial neural nets are of interest primarily because they may be able to emulate the speed and performance of real biological neural nets using many simple slow computational elements operating in parallel. They thus offer one possible solution to the problem of obtaining the massive parallelism and computational requirements that are presumed to be required for such problems as speech recognition.

Two neural nets are logically required to implement an optimum classifier for binary patterns. One net forms the weighted sums to calculate quantities related to the likelihood of the different classes and the second picks the maximum.

A net that forms weighted sums is presented in Figure 3. The topology of this net is similar to that of a perceptron¹². An input pattern \mathbf{x} is applied at the bottom of this net and an output pattern \mathbf{y} is produced at the top. The first layer of nodes sends values of the input pattern to the links feeding the second layer. The second layer of nodes uses nonlinear threshold logic elements¹⁰ to sum weighted values of the inputs and add internal offsets. Output values from the second layer are

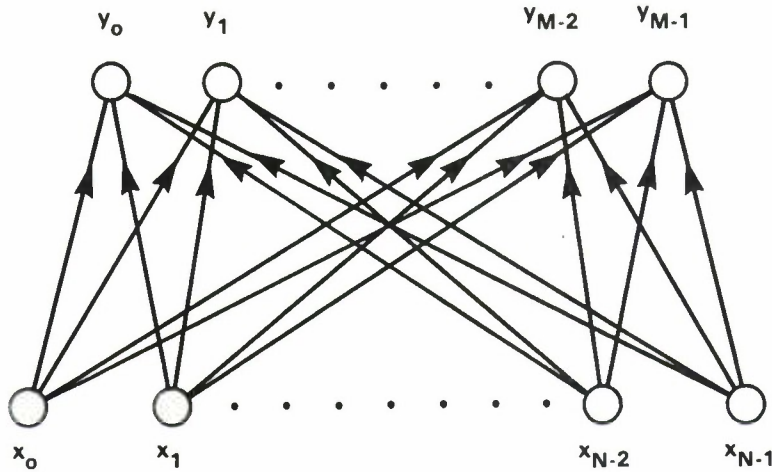
$$y_j = f(c_j + \sum_{i=0}^{N-1} w_{ij}x_i), \quad (8a)$$

where

$$f(\alpha) = \begin{cases} \alpha & \text{if } \alpha > 0 \\ 0 & \text{if } \alpha \leq 0 \end{cases} \quad (8b)$$

In these equations, $f(\alpha)$ is a nonlinear function that models the nonlinearity inherent in a biological neuron[†], c_j is an internal offset associated with each threshold logic node, and w_{ij} are positive or negative weights associated with the links. The internal offsets and weights are selected differently for the three different situations described in the previous section. A binary symmetric

[†] Biological neurons saturate for large enough α . In this discussion we are interested in the monotonically increasing portion of the input-output characteristic.



77677-3

Figure 3. Feed-forward perceptron neural net used to calculate values related to the likelihood of each of M pattern classes for patterns with N elements. All nodes are analog-threshold logic nodes with internal thresholds set to zero. Weights depend on the stored states.

channel with $\rho = \epsilon$ requires the weights and offset in Equation (5) if elements of the input vector take on the values $+1$ and -1 , and the weights and offsets in Equation (6) if elements take on the values 0 and 1 . In the more general case when $\rho \neq \epsilon$, and the inputs take on the values 0 and 1 , the weights and offsets are as in Equation (7). The resultant output values y_j are $N - N_{\text{ham}}^j$ for the binary symmetric channel, and L_j for the general case.

A number of different nets can be used to pick the maximum value from the y_j outputs of the perceptron net. In situations where it is only important to know when the input matches a stored state very closely, it is sufficient to identify those second-level nodes in Figure 3 with output values that exceed a specified threshold. This can be performed by modifying the constant added in (8a) such that only the output of those nodes corresponding to closely matching stored states are positive. For example, for the binary symmetric channel with $+1$ and -1 inputs, if c_j in (8a) is changed to $\Delta - \frac{N}{2}$ then only nodes corresponding to exemplars with a Hamming distance less than Δ from the input will have positive outputs.

In the more general situation, a net must select the maximum over the M y_j values. We have developed three topologically different neural net structures which perform this task. These nets maintain the highly parallel structure necessary to achieve the theoretical computation speed-up provided by multiple processors. They could thus be used in a larger system when their outputs feed other nets without compromising the overall computational speed of the system. One feed-forward net uses a brute force approach to perform binary comparisons between all input values. A second feed-forward net uses a binary tree to reduce the number of nodes required. Finally, a third fully-connected net sometimes called a "winner-take-all" net uses strong inhibition between nodes and iterates until the maximum is found.

A brute force feed-forward net that picks the maximum from eight inputs is presented in Figure 4. The inputs labelled y_0 through y_7 are on the bottom of the net, and the outputs labelled z_0 through z_7 are along the left diagonal. This network is designed such that only the output corresponding to the maximum input will be positive. The filled circles in this net represent hard-limit nodes that compare the values of all inputs. Hard limit nodes are similar to threshold logic nodes (Equations 8a and 8b) except the function f is defined by

$$f(\alpha) = \begin{cases} 1 & \text{if } \alpha > 0 \\ 0 & \text{if } \alpha \leq 0. \end{cases} \quad (9a)$$

These nodes are simpler to implement than threshold-logic nodes because linearity above threshold is not required.

The outputs, z_j , in Figure 4 are

$$z_j = f \left[\sum_{i>j} f(y_j - y_i) - \sum_{i<j} f(y_i - y_j) + j - (M - 1.5) \right] \quad , \quad (9b)$$

where f is as in Equation (9a) and $0 \leq i, j < M$.

Hard limit nodes in Figure 4 perform binary comparisons required in Equation (9b) between all inputs. Internal thresholds in the output nodes and weights are set such that an output node is positive only when the results of all binary comparisons with the associated input indicate that that input is greatest. A limitation of this net is that it becomes very large for large M because it requires $O(M^2)$ nodes to pick the maximum of M inputs. For example, 5050 nodes are required to pick the maximum of 100 inputs.

A neural net that is more efficient for large numbers of inputs can be built using the two-input comparator subnet presented in Figure 5. This subnet produces logical outputs (z_0, z_1) indicating which input was maximum and also one analog output (\max) equal to the maximum value. Whenever the inputs (y_0 and y_1) differ, only the logical output corresponding to the maximum input will be nonzero, and the value of \max will be that of the maximum input. The comparator subnet uses threshold logic nodes represented by open circles in Figure 5, and hard-limit nodes represented by filled circles. In addition, all internal offsets (c_j in Equation (8a)) in Figure 5 are zero.

A network that picks the maximum of M inputs can be built using comparator subnets arranged in a layered binary tree. Such a net includes $M-1$ comparator subnets arranged in roughly $\log_2 M$ layers when the maximum of M inputs must be selected. For example, it requires only 594 nodes to pick the maximum from 100 inputs. An example of such a net that picks the maximum of eight inputs is presented in Figure 6. The inputs in this net are at the bottom, and the outputs are at the top. Four comparator subnets in the first layer, two comparator subnets in the second layer, and one partial subnet in the third layer from the bottom of this net are used to find the maximum input. The maximum value is fed forward from the input through the threshold-logic nodes (open circles) to the final partial subnet. The decisions concerning which input was maximum are fed forward from the input through the hard-limit nodes (filled circles) to the output

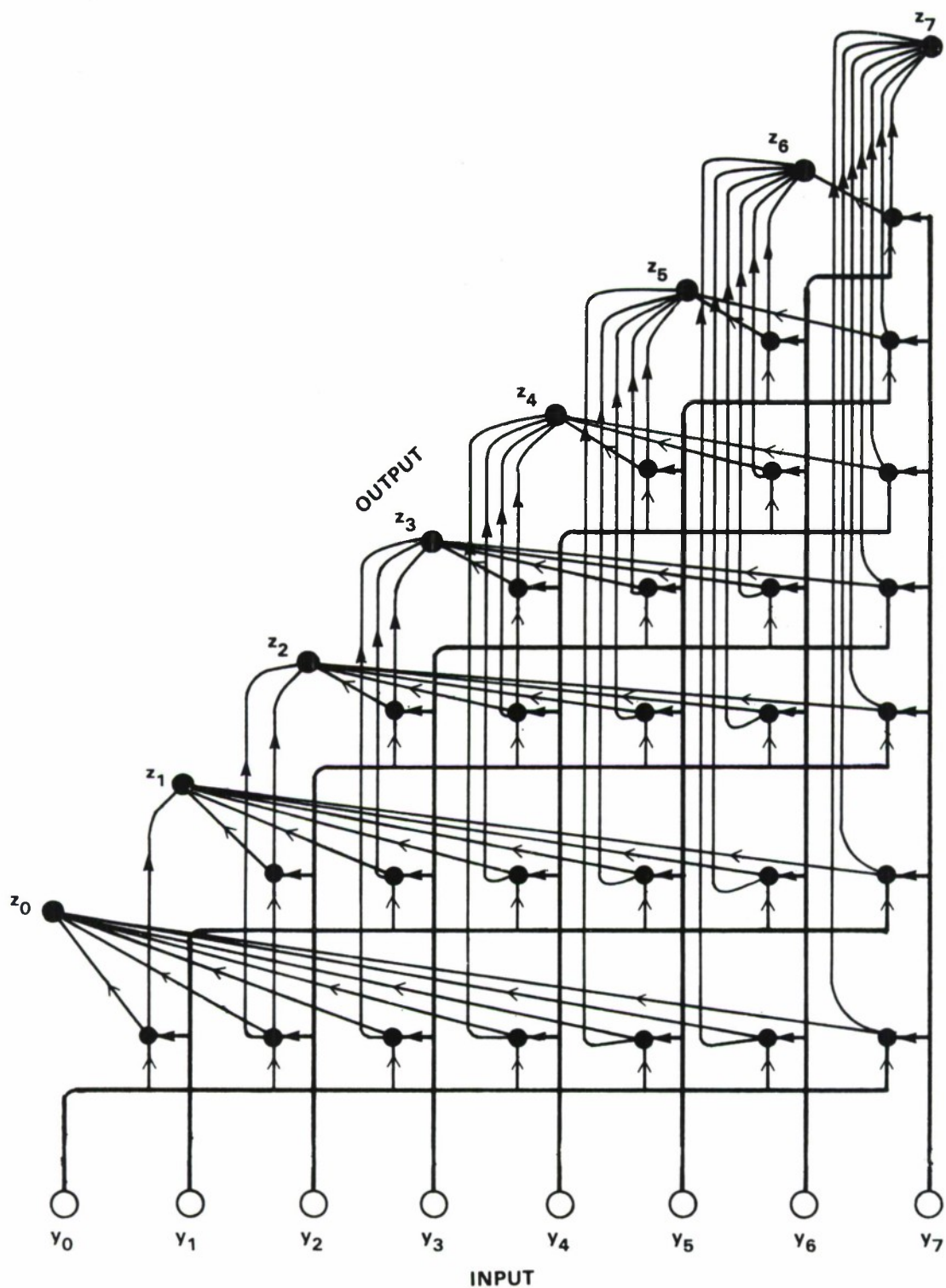


Figure 4. Feed-forward neural net that determines which of eight inputs is maximum by explicitly performing all binary comparisons. Nodes denoted by filled circles are hard-limit nodes and nodes denoted by open circles are analog-threshold logic nodes. Internal thresholds on all nodes except the output nodes are zero. Internal thresholds on nodes $z_0, z_1, \dots, z_6, z_7$ are $-6.5, -5.5, \dots, -0.5, +0.5$, respectively. Weights are either $+1$ (open arrows) or -1 (filled arrows).

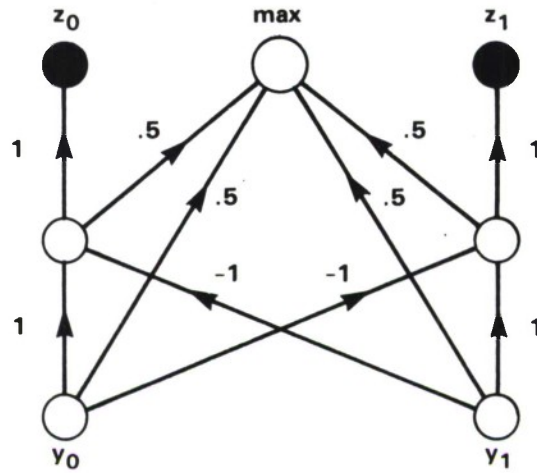


Figure 5. Comparator subnet that selects the maximum of two inputs. Internal thresholds on both hard-limit nodes (filled circles) and analog-threshold logic nodes (open circles) are zero.

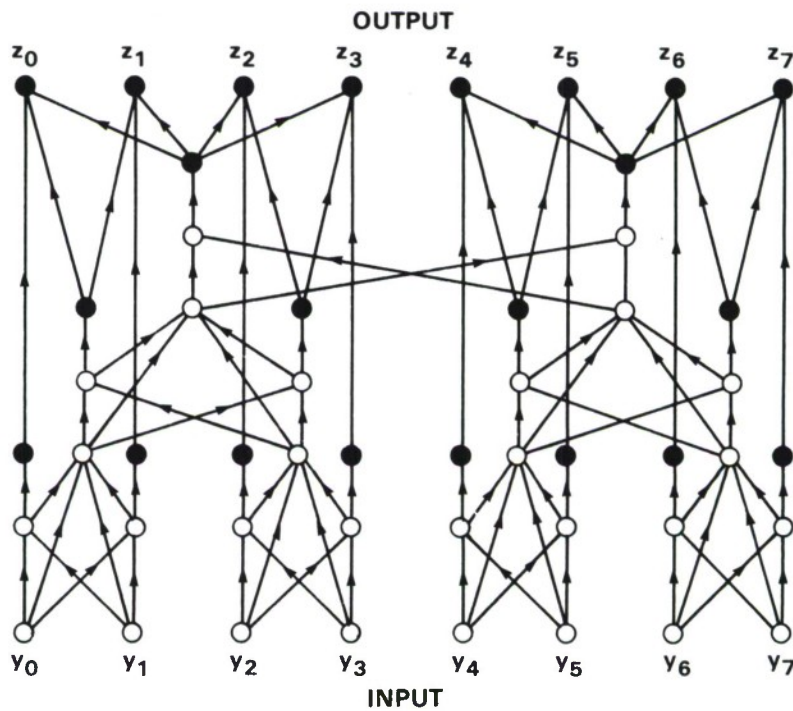


Figure 6. Feed-forward neural net that determines which of eight inputs is maximum using a binary tree and comparator subnets from Figure 5. Internal thresholds on both hard-limit nodes (filled circles) and analog-threshold logic nodes (open circles) are zero except for the output nodes. Internal thresholds on nodes $z_0, z_1, \dots, z_6, z_7$ are -2.5 . Weights for all comparator subnets in this net are as in Figure 5. All other weights are $+1$.

nodes. After the input propagates to the output, only that output corresponding to the maximum input will be high.

The above two nets use strictly feed-forward connections and are relatively large. A less complex net that uses feedback connections and will be referred to as a maxnet, is presented in Figure 7. This net is motivated by the large numbers of connections in biological neural nets and by laterally interconnected networks described by Kohonen¹². Although this net is similar in structure to the Hopfield net¹, it uses threshold-logic nodes instead of hard-limit nodes and feeds the output of each node back to its input instead of disallowing this feedback path. The maxnet is a fully connected net made up of only M threshold logic nodes with internal thresholds set to zero. Input values are applied at time zero through the input nodes on the bottom of Figure 7. This initializes node outputs for each node at time zero ($\mu_j(0)$) to the input values:

$$\mu_j(0) = y_j, j = 0, 1, \dots, M-2, M-1 \quad (10a)$$

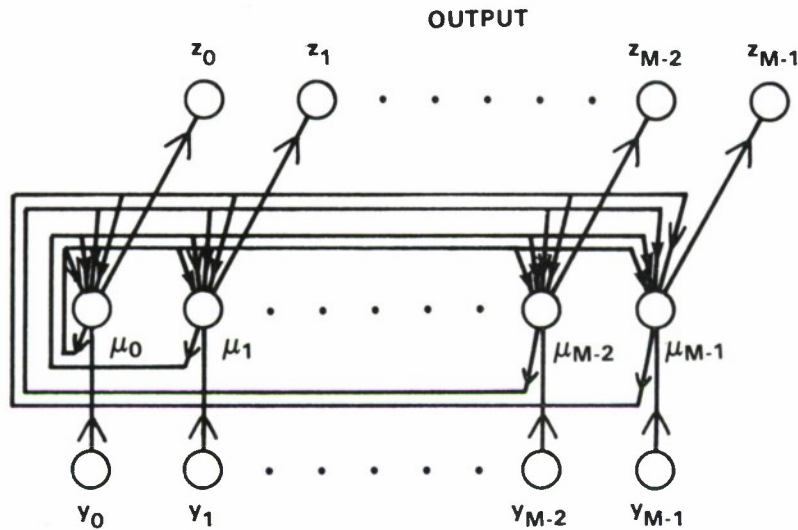


Figure 7. Iterative neural net denoted "maxnet" that determines which of M inputs is the maximum. The inputs are applied prior to time zero and then removed, and the outputs are valid after the network converges. All nodes are analog-threshold logic nodes with internal thresholds equal to zero. Each node connects to itself and all other nodes. Weights are $-w$ (solid arrows) or $+1$ (open arrows).

The network then iterates to find the maximum via the following equation:

$$\mu_j(t+1) = f[\mu_j(t) - \sum_{i \neq j} w_{ij} \mu_i(t)] \quad (10b)$$

In this equation f is the threshold logic function described in Equation (8b) and w_{ij} is the inhibitory weight between nodes. Each node inhibits all other nodes with a value equal to the node's output multiplied by a small negative weight. Each node also feeds back to itself with unity gain.

After convergence, only that output node corresponding to the maximum input will have a non-zero value. This value will, in general, be less than the original, time zero, value of that node. The output values of the net are thus simply the node output values after convergence:

$$z_j = \mu_j(\infty), j = 0, 1, \dots, M-2, M-1. \quad (10c)$$

The maxnet net will converge and find the maximum input when

$$w_{ij} = w < \frac{1}{M-1}, \quad (10d)$$

where, by convergence we mean the node outputs stop changing and only the output of one node corresponding to the maximum input is positive.

The proof of convergence depends primarily on the fact that the inhibition to the node containing the maximum value is always less than the inhibition to other nodes, and that at convergence the inhibition to the node with the maximum value reduces to zero. Denote the total inhibition in Equation (10b) from all other nodes to node j as inhib_j where:

$$\text{inhib}_j(t) = \sum_{i \neq j} w_{ij} \mu_i(t). \quad (11)$$

If node j^* corresponds to the maximum input, then on the first iteration $\text{inhib}_{j^*}(1)$ will be less than the inhibition to all other nodes. This follows because all node outputs are positive and the sum of all outputs, excluding one in Equation (11), will be minimum when the maximum is excluded. Node j^* will thus remain the maximum after the first iteration. By induction, it will remain the maximum over all iterations.

The remainder of the proof of convergence depends on demonstrating that the output of node j^* is never driven to zero but the outputs of all other nodes are. When Equation (10d) is satisfied, $\text{inhib}_j(t)$ is always less than the average value of all other node outputs. The inhibition to node j^* will thus be less than the average of the output of all nodes. Whenever a maximum exists, this inhibition will always be less than the current output of node j^* because the maximum of a set of positive numbers is always greater than the average. The output of node j^* will thus not be driven to zero while any other nodes have non-zero outputs. After all other node outputs are driven to zero, the inhibition to node j^* drops to zero, and the output of j^* remains constant. The outputs of all other nodes will always be driven to zero because the inhibition to these nodes remains positive on all iterations and approaches a positive constant as time increases. In practice, the maxnet will still converge and find the maximum when each weight w_{ij} is set to $\frac{1}{M-1}$ plus a small random component. This forces the net to find a maximum when the inputs to all nodes are identical.

The behavior of the maxnet is illustrated in Figure 8. This figure presents the node outputs for a 10-node maxnet at iterations 0 through 3 when the net converged. The initial values to the net come from the perceptron likelihood calculation net presented in Figure 3 when the number

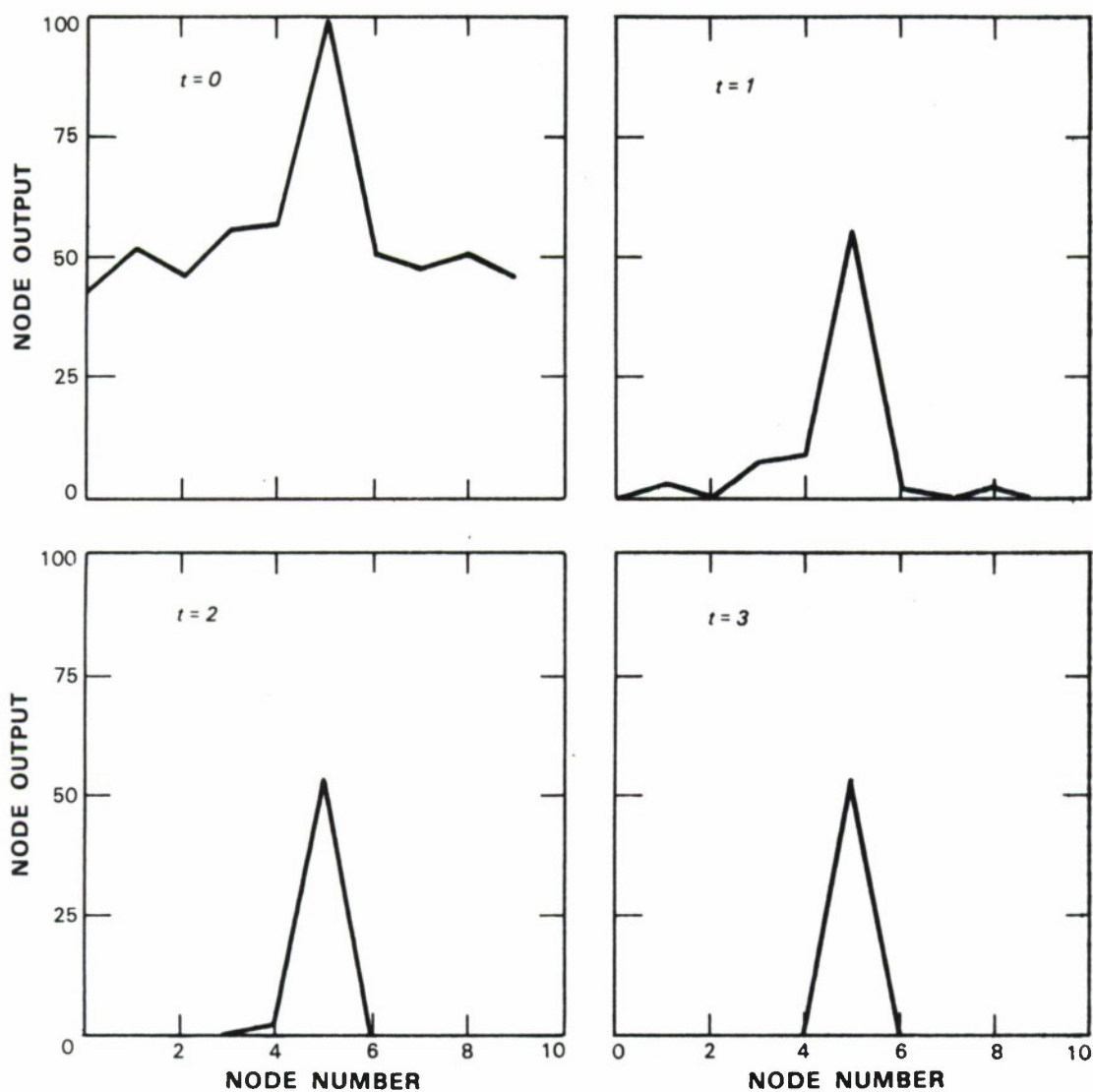


Figure 8. Node outputs in a 10-node maxnet at iterations zero through 3 when the net converged. Initial values come from the perceptron net presented in Figure 3 when the number of classes (M) is 10, the number of input nodes (N) is 100, and the input to the perceptron net is the exemplar pattern for class number 5.

of classes (M) is 10, the number of input nodes (N) is 100, and the input to the classifier is the exemplar pattern for class number 5. Exemplars were selected by randomly setting each element in every exemplar to +1 or -1 with equal probability. The output values from the maxnet nodes range from zero to 100 which is the number of input nodes in the classifier. At time zero, node 5 has a value of 100 because the input pattern exactly matches the exemplar for class 5. All other nodes have values from a binomial distribution with a mean of $\frac{N}{2} = 50$ and a standard deviation of $\sqrt{.25N} = 5$. This occurs because the Hamming distance has a binomial distribution when exemplars are selected at random as described above. After the first iteration, output values are reduced by the average of all nodes at time zero and the outputs of six nodes remain positive. Output values are then reduced further on the second iteration where only three outputs remain positive. Finally, only the maximum output remains positive after the third iteration.

A similar example is presented in Figure 9 for a maxnet with 100 instead of 10 nodes where the number of input nodes to the perceptron likelihood calculation net is 1000. As can be seen, the number of iterations required for convergence increases only slightly from 3 to 9. This increase is primarily caused by the greater number of nodes with values at the high end of the binomial distribution when there are 100 nodes.

Convergence is slower when the peak value across nodes in the maxnet is less distinct. This is illustrated in Figure 10. This figure is similar to Figure 9 except the input to the perceptron net was passed through a noisy binary symmetric channel where the probability of changing a bit (ρ) was set to 0.4. The initial value of node 50 is roughly 600 because only roughly 20% of the bits in the input match the exemplar for class 50. Other nodes still have a binomial distribution with mean 500 and standard deviation of $\sqrt{.25N} = 15.8$. As can be seen, the network converges in 27 iterations. Similar experiments were performed when the number of classes (M) in the classifier ranged from 2 to 100, the number of elements in the patterns (N) was set to $10M$, and the probability of error in the binary symmetric channel (ρ) ranged from 0.0 to 0.5. Results are presented in Figure 11.

Figure 11 presents the average number of iterations until convergence for the maxnet of Figure 7 versus the probability of error in the binary symmetric channel for the above cases. It was obtained from Monte Carlo experiments examining 100 different runs per point. As can be seen, the average number of iterations required for convergence is small (<10) for as many as 100 classes when there is a well-defined peak and the probability of error is less than 0.1. The average number of iterations also does not grow strongly with the number of nodes in the maxnet. The average number of iterations required for convergence rises gradually to be less than 20 when the probability of error is 0.3. Above this level, the average number of iterations rises to a value that is roughly 10% above M when the probability of error is 0.5. These results demonstrate the utility and robustness of the maxnet. The net always converges and finds the node with the maximum value, and the number of iterations required for convergence does not grow rapidly as the number of classes increases. Furthermore, the number of iterations increases only when the error rate of the classifier is large and the utility of the classifier itself is questionable.

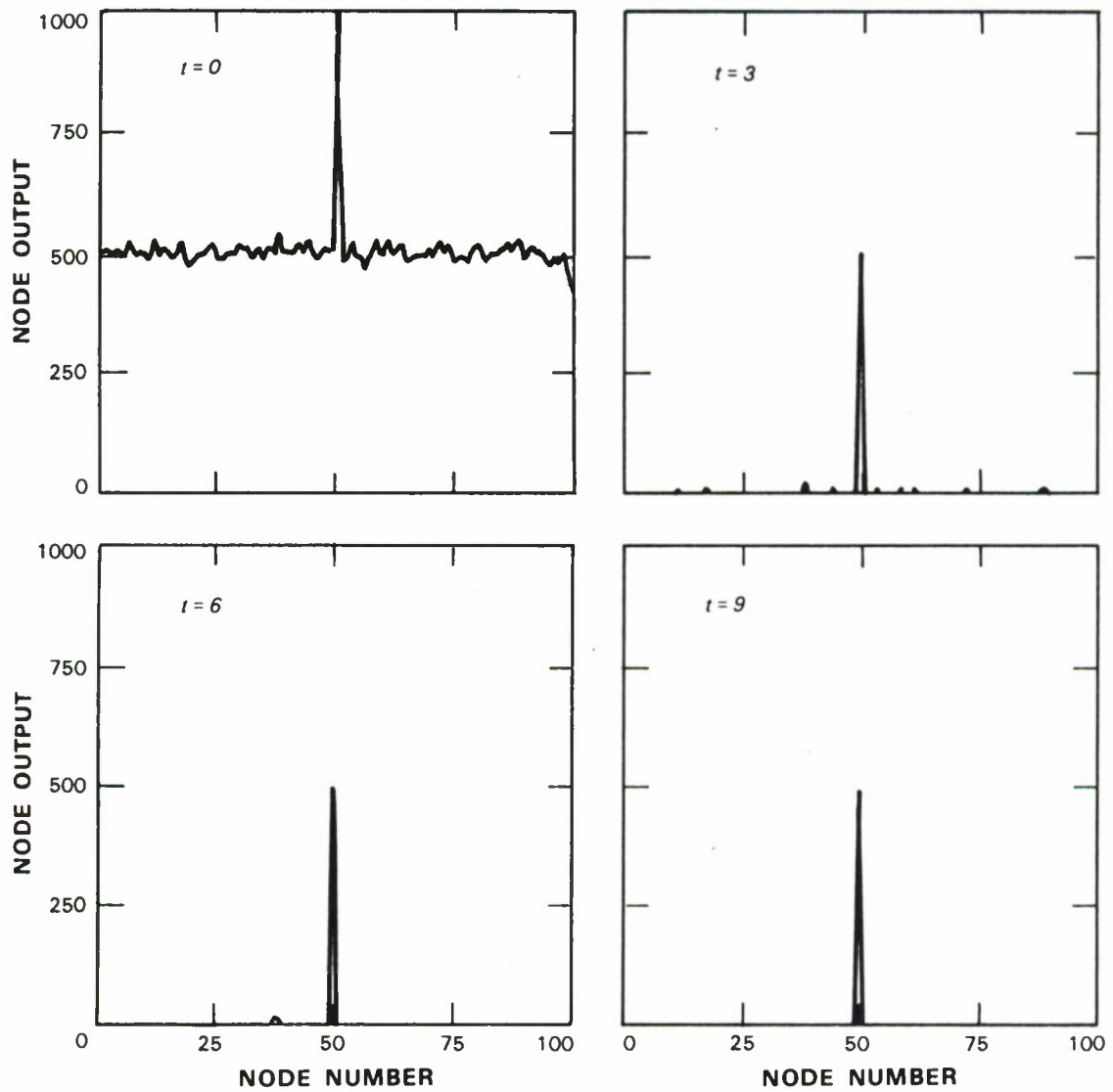


Figure 9. Node outputs in a 100-node maxnet at iterations zero through 9 when the net converged. Initial values come from the perceptron net presented in Figure 3 when the number of classes (M) is 100, the number of input nodes (N) is 1000, and the input to the classifier is the exemplar pattern for class number 50.

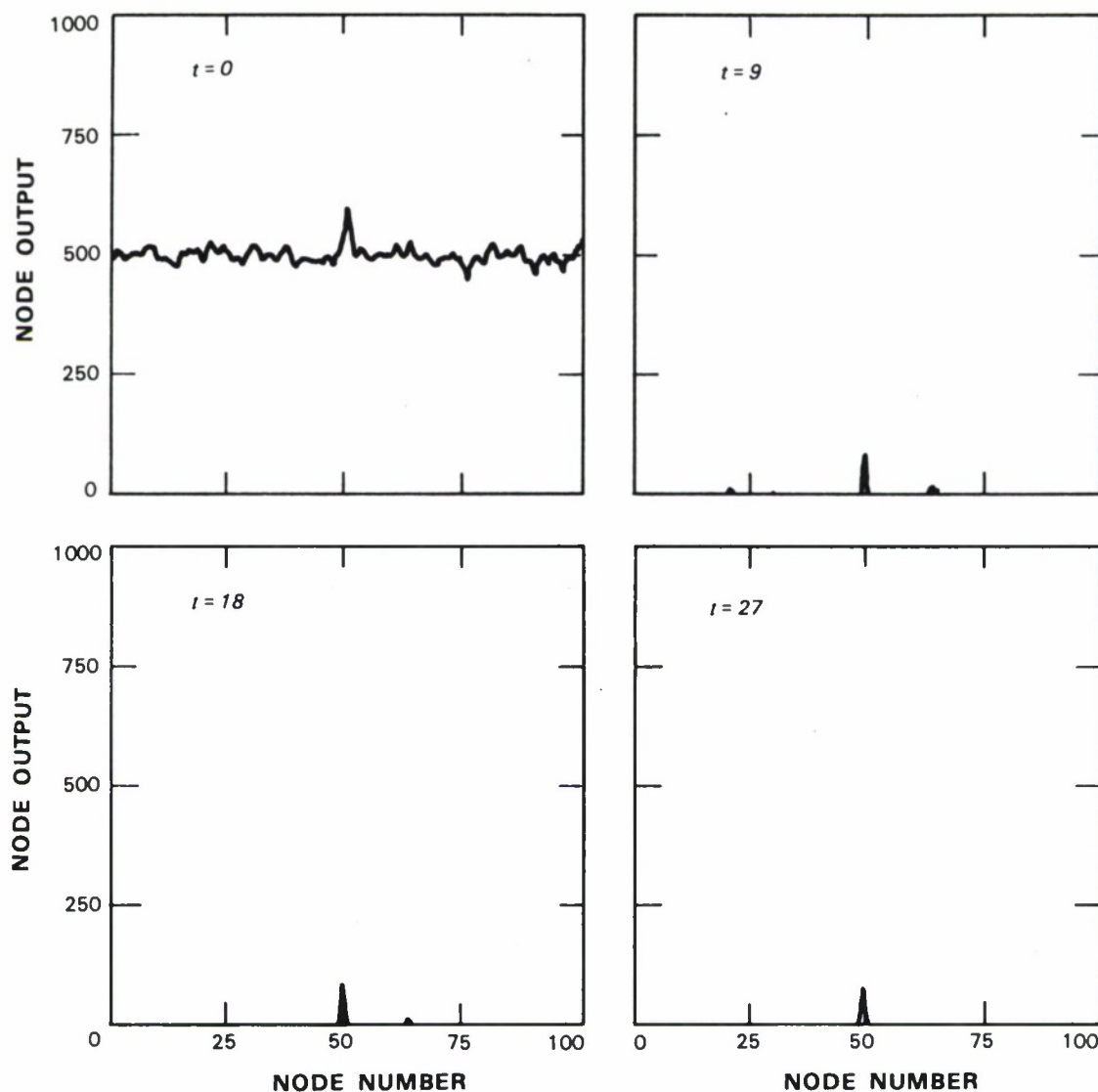


Figure 10. Node outputs in the 100-node maxnet of Figure 9 when the input to the classifier is the exemplar pattern for class number 50 after being passed through a memoryless binary-symmetric channel with an error probability, p , of 0.4.

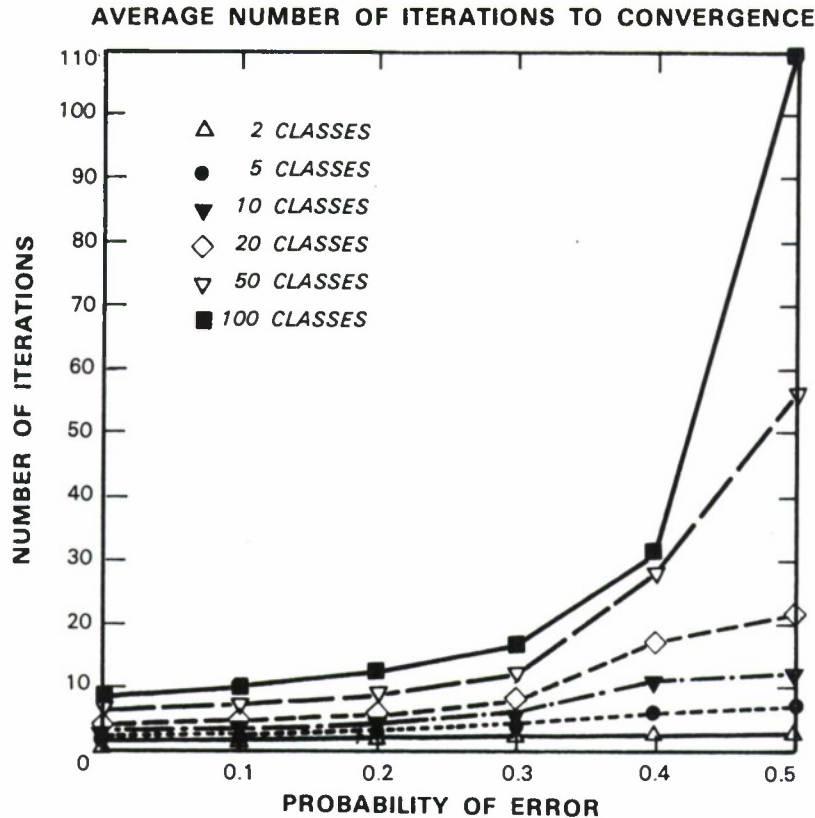


Figure 11. Average number of iterations until convergence for the maxnet of Figure 7. Initial values come from the perceptron net presented in Figure 3 when the number of classes (M) ranges from 2 to 100, the number of input nodes (N) is $10M$, and the input is the exemplar for one pattern after being passed through a binary-symmetric channel with the specified probability of error.

A comparison of the three different types of nets described above for picking the maximum is presented in Table I. This table illustrates that the maxnet in Figure 7 requires the fewest nodes to pick the maximum value. The brute force feed-forward net in Figure 4 becomes intractable for large numbers of inputs because the number of nodes in this net grows $O(N^2)$. The binary tree net in Figure 6 is more reasonable; however, it still requires roughly six times the number of nodes used in the maxnet. The maxnet should thus be preferred whenever the deciding factor is the number of nodes required and the slight delay added by the need to iterate is acceptable. For large numbers of inputs, ($M > 10$) the number of interconnects required is always greatest for the brute force net and the maxnet and least for the binary tree. The binary tree may thus be preferred when the deciding factor is the number of interconnects. In the following, we use the maxnet.

TABLE I Comparison of Three Neural Nets That Pick The Maximum of M Inputs (Node Counts Do Not Include Input Or Output Nodes)					
Net	Structure	Types of Nodes Required	Number of Nodes Required		
			M = 10	M = 100	M = 1000
Brute Force	Feed-Forward	Hard-Limit	55	5,050	500,500
Binary Tree	Feed-Forward	Hard-Limit and Threshold-Logic	54	594	5,994
Maxnet	Fully-Connected, Iterate to Convergence	Threshold-Logic	10	100	1,000

A block diagram of a complete neural net classifier made up of a perceptron likelihood calculator and a maxnet is presented in Figure 12. This complete classifier will be referred to as a Hamming net. The Hamming net is an efficient optimum classifier made up of only threshold-logic nodes and interconnects. It is operated by presenting a binary pattern at the input for at least the time it takes the input to propagate to the maxnet nodes, by removing the input, and then by waiting until the maxnet converges. After convergence, only the output node corresponding to the selected class will be positive.

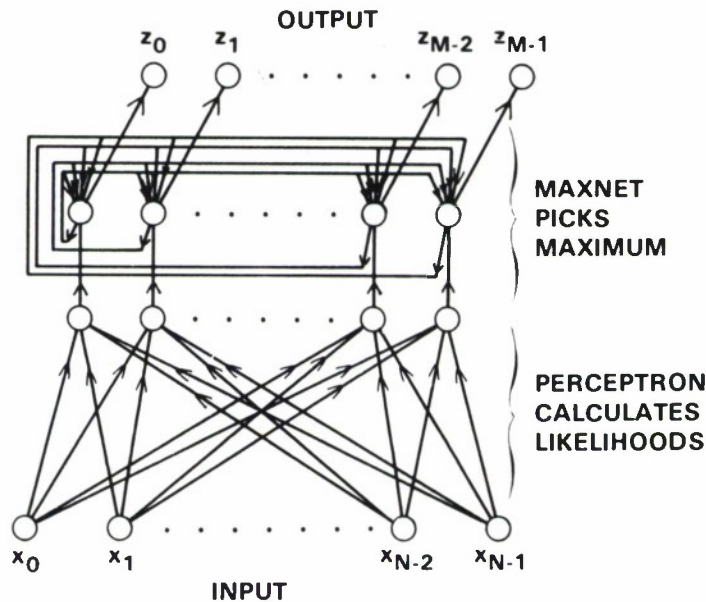


Figure 12. Complete optimum neural-net classifier referred to as a Hamming net made up of a perceptron to calculate likelihoods, and a maxnet to select the node with the maximum likelihood.

4. THE HOPFIELD CLASSIFIER

A highly connected neural net was described by Hopfield in References 1 to 3 that can be used as an associative memory. The net described in¹ that uses hard-limit nodes is presented in Figure 13. Input values are applied at time zero to these nodes through the bottom threshold-logic nodes. This initializes node outputs for each node at time zero ($\mu_i(0)$) to the input values

$$\mu_i(0) = x_i, \quad i = 0, 1, \dots, N-2, N-1 \quad . \quad (12a)$$

The network then iterates using the following equation:

$$\mu_j(t+1) = f\left(\sum_{i=0}^{N-1} t_{ij}\mu_i(t)\right) \quad . \quad (12b)$$

In this equation f is a modified hard-limit function and t_{ij} is weight applied to the output of node i that feeds to node j . If we assume the elements of the input vector x take on values $+1$ and -1 , respectively, for the $+1$ and -1 states, then f is the symmetric hard-limit function

$$f(\alpha) = \begin{cases} 1 & \text{if } \alpha > 0 \\ -1 & \text{if } \alpha \leq 0 \end{cases} \quad , \quad (12c)$$

and the weights are specified by

$$t_{ij} = \sum_{s=0}^{M-1} x_i^s x_j^s \quad , \quad i \neq j \quad , \quad (13a)$$

and

$$t_{ij} = 0, \quad i = j \quad , \quad (13b)$$

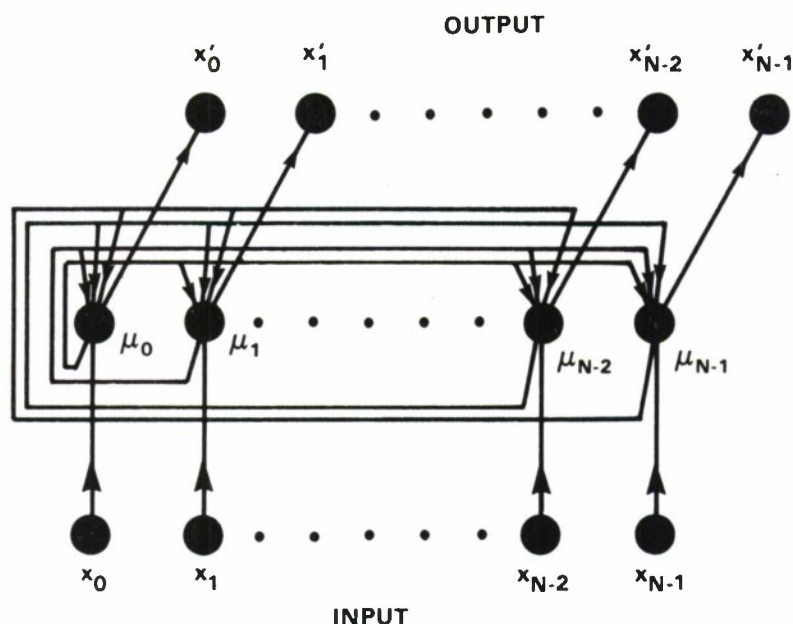
where x_i^j is element i of the exemplar for pattern j . The output of each node is fed to every other node with a weight that is symmetric, and each node does not feedback to itself. After convergence, the output of the net is the final pattern represented by the outputs of the nodes

$$x_i' = \mu_i(\infty), \quad i = 0, 1, \dots, N-2, N-1 \quad . \quad (14)$$

Hopfield¹ first demonstrated that when this net is trained with M exemplar patterns using Equation (13), and an exemplar is presented at time zero, then the final pattern in the net after convergence will be one of the exemplars with high probability if

$$M < .15N \quad . \quad (15)$$

The exemplars thus form stable states of the net. Hopfield's statistical results were obtained with randomly generated exemplars. It is possible and relatively easy to select a set of M exemplars that satisfy Equation (15) but do not form stable states in the Hopfield net. These exemplars must have many elements in common. When an exemplar for one of these patterns is presented at time zero, the network doesn't converge to any of the trained exemplars. Instead it converges



77677-13

Figure 13. Iterative Hopfield neural net. The inputs are applied prior to time zero and then removed, and the outputs are valid after the network converges. Nodes denoted by filled circles are symmetric hard-limit nodes. Internal thresholds in all nodes are set to zero. Each node in the middle row connects to all other nodes but not to itself. Weights are specified by t_{ij} (solid arrows) or are $+1$ (open arrows).

to a spurious pattern never seen before. This problem of spurious states also occurs when a noisy exemplar is presented to the net. Even when the M exemplars are stable states of the net, there is no guarantee that noisy versions of these exemplars passed through discrete memoryless channels and presented at time zero will converge to the original exemplar. Hopfield¹, for example, observed that the number of spurious states found increases substantially as more and more elements in the input exemplar are corrupted.

The Hopfield neural net can be used as a classifier only when: (1) the exemplars for the patterns to be classified form stable states and converge to themselves when presented at time zero as input, and (2) a mechanism is provided to determine which of the M exemplars the net is closest to after convergence. The first requirement is a necessary condition for proper operation. The second is necessary because the Hopfield net by itself is not a neural-net classifier. It is more like a preprocessor which still requires a classification net to select which of M exemplars a pattern is closest to.

It is difficult to satisfy the requirement that exemplars form stable states without actually running the Hopfield net. In general, patterns that are more random will satisfy this requirement more easily than patterns with many bits in common. We have satisfied this requirement by selecting patterns carefully using trial and error procedures, by using randomly generated patterns where the Hamming distances between all patterns were within certain bounds, and by using an orthogonalization technique described in⁶.

The orthogonalization technique involves generating patterns \mathbf{b}^k that are orthogonal to the exemplars \mathbf{x}^s unless $k = s$. The weights in the net are then given by

$$t_{ij} = \sum_{s=0}^{M-1} x_i^s b_j^s, \quad i \neq j, \quad (16a)$$

and

$$t_{ij} = 0, \quad i = j. \quad (16b)$$

Equation (16) then replaces Equation (13) as the recipe for determining weights, and the operation of the net is otherwise the same as without orthogonalization. The \mathbf{b}^k patterns are found from

$$\mathbf{B} = \mathbf{X} \cdot \mathbf{C}^{-1} \quad (17a)$$

In this equation \mathbf{B} is an N by M array where each row is the orthogonal pattern \mathbf{b}^i , \mathbf{X} is an N by M array where each row is the exemplar \mathbf{x}^i , and \mathbf{C} is an M by M correlation matrix for the exemplars:

$$c_{ij} = \sum_{k=0}^{N-1} x_k^i x_k^j. \quad (17b)$$

The inverse in Equation (17a) will exist provided the \mathbf{x}^s exemplars are linearly independent. When the exemplars are not linearly dependent, the more general Moore-Penrose matrix inversion technique¹³ can be used.

The second requirement for using the Hopfield net as a classifier can be satisfied in two ways. First, if a “no-match” output is allowed for spurious states, the final pattern in the net need only be compared to each of the exemplars. In this case, a perfect match results in an output, otherwise the output indicates a “no-match” condition. A net that performs this type of classification is presented in Figure 14. Comparisons between the final state of the Hopfield net and exemplars are performed using a perceptron with M nodes. The weights in the perceptron from internal node x_i^j to node z_j are simply the elements from exemplar j :

$$w_{ij} = x_i^j \quad (18)$$

Internal offsets in the output nodes are set to $\epsilon - N$ where $\epsilon < 1$. The output of output node j will thus be non-zero only when the final pattern in the Hopfield net exactly matches the exemplar for pattern j .

Another technique for using the Hopfield net in a classifier eliminates the possibility of a “no-match” output. Here, the Hamming distance between the final pattern in the net and exemplars for all patterns is computed and the pattern with the minimum Hamming distance is selected. This can be performed with the Hamming net classifier presented in Figure 12 when the input to the Hamming net is the final state or output of the Hopfield net. A classifier that uses

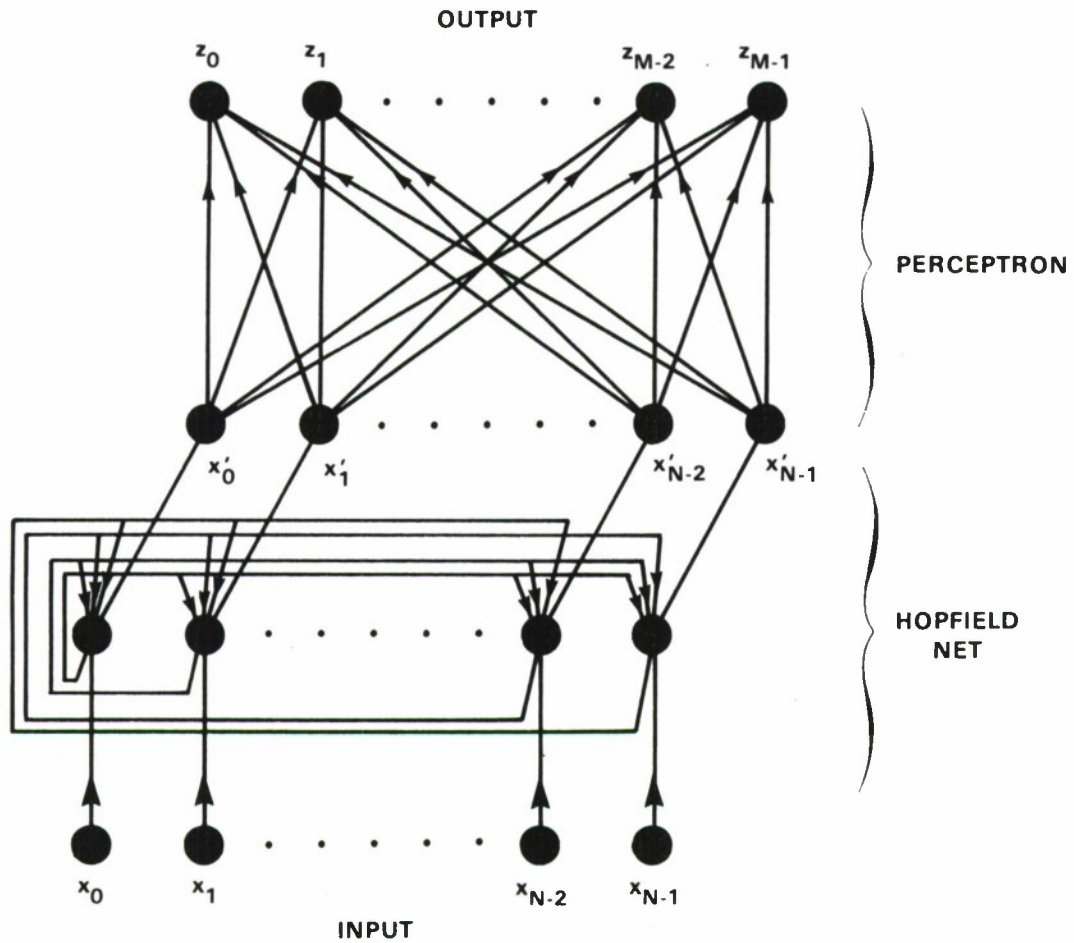


Figure 14. Complete Hopfield neural-net classifier made up from a Hopfield net followed by a perceptron. The Hopfield net is as in Figure 13. The perceptron is designed as in Figure 3 except all nodes are hard-limit nodes and the internal thresholds in the final output nodes are set to $\epsilon - N$ instead of zero, where $\epsilon < 1$.

this technique will be called a Hopfield-Hamming classifier. The Hopfield-Hamming classifier uses the Hopfield net as a preprocessor for the Hamming net. Since the Hamming net is optimal, the Hopfield-Hamming net is generally suboptimal and always requires more nodes than the Hamming net.

Table II compares the two classifiers that use the binary Hopfield net to the optimal Hamming net classifier. As can be seen, the two classifiers that use Hopfield nets require more nodes and many more interconnects than the optimal Hamming net for the normal situation when the number of classes is less than the number of elements in each input pattern. For the examples in the table, classifiers using the Hopfield net require almost an order-of-magnitude more interconnects than the Hamming net and more nodes than the Hamming net. This is because the Hopfield net requires $O(N^2)$ interconnects while the Hamming net requires only $O(M^2)$ interconnects and in the examples $N = 10M$. In general, classifiers using the Hopfield net always require nearly an order of magnitude more interconnects than the Hamming net because Hopfield¹ requires $M < .15N$. When orthogonalization is used as in⁶ and $N = M$, the simpler Hopfield classifier is roughly equivalent in complexity to the Hamming net while the more complex Hopfield-Hamming net requires more nodes and almost twice as many interconnects as the Hamming net. The more complex Hopfield-Hamming net is probably to be preferred over the simpler Hopfield classifier, however, because it does not allow a “no-match” output to occur when the Hopfield net converges to a spurious state.

TABLE II Comparison of Three Different Neural Nets That Can Be Used To Classify M Binary Patterns When Each Pattern Has N Elements (Counts Of Numbers Of Nodes And Interconnects Do Not Include Input And Output Nodes Or Interconnects Between Major Internal Subnets)				
Net	Description	Nodes	Interconnects	Nodes/Interconnects M=10, N=100
Hopfield	Hopfield Net Followed By Perceptron	$2N+M$	$N^2+N(M-1)$	210/10,900
Hopfield- Hamming	Hopfield Net Followed By Hamming Net	$2N+2M$	$N^2+M^2+N(M-1)$	220/11,000
Hamming	Perceptron Followed By Maxnet	$N+2M$	M^2+NM	120/1,100

5. COMPARISONS BETWEEN HOPFIELD CLASSIFIERS AND THE HAMMING NET

Two experiments were performed using 8 patterns with 120 elements each, to compare the behavior of a Hamming net classifier, a Hopfield-Hamming classifier, and a Hopfield classifier. The first experiment used the set of visually recognizable handcrafted digit patterns shown in Figure 15. These patterns represent seven digits plus one block pattern selected because it had a large Hamming distance to the others. The second experiment, used random patterns where each element was +1 or -1 with a probability of 0.5. In this experiment a pattern was used if, and only if, its Hamming distance from the already accepted exemplars fell in the range 55 to 65. Tables III and IV list the Hamming distances for the 8 exemplars (s0 through s7) used in the two experiments. All exemplar patterns were stable; i.e., any exemplar given as the input to a network did not change as a result of iterating. In both experiments each of the 8 exemplars was randomly perturbed 40 times at a specified error rate using a binary symmetric channel and applied as the input to each of the three classifiers. This provided 320 runs per data point.

The results of the first experiment using digit patterns are presented in Figure 16a. Percent correct in this and other figures was computed by counting a trial as correct only when the correct input pattern was selected. It can be seen that the Hopfield net fails immediately at an error rate of .025 and that the Hopfield-Hamming net begins to fail at the error rate of .150, whereas

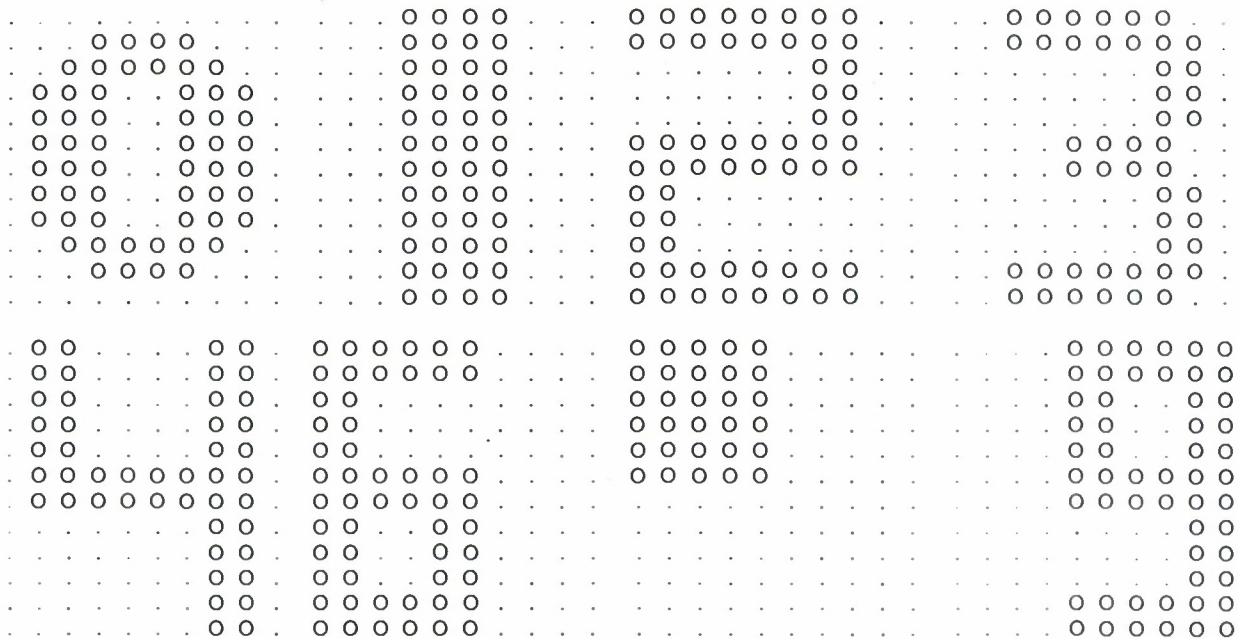


Figure 15. Digit patterns used in Experiment No. 1.

TABLE III								
Hamming Distances Between Digit Patterns Used In Experiment No. 1								
	s0	s1	s2	s3	s4	s5	s6	s7
s0	0	48	64	58	48	74	58	74
s1	48	0	54	50	78	54	54	54
s2	64	54	0	36	56	30	60	66
s3	58	50	36	0	36	60	62	36
s4	48	78	56	36	0	66	48	52
s5	74	54	30	60	66	0	42	84
s6	58	54	60	62	48	42	0	72
s7	74	54	66	36	52	84	72	0

TABLE IV								
Hamming Distances Between Random Patterns Used In Experiment No. 2								
	s0	s1	s2	s3	s4	s5	s6	s7
s0	0	62	59	61	57	59	64	63
s1	62	0	65	57	63	61	64	59
s2	59	65	0	62	58	58	55	58
s3	61	57	62	0	60	58	57	58
s4	57	63	58	60	0	56	65	56
s5	59	61	58	58	56	0	65	62
s6	64	64	55	57	65	65	0	61
s7	63	59	58	58	56	62	61	0

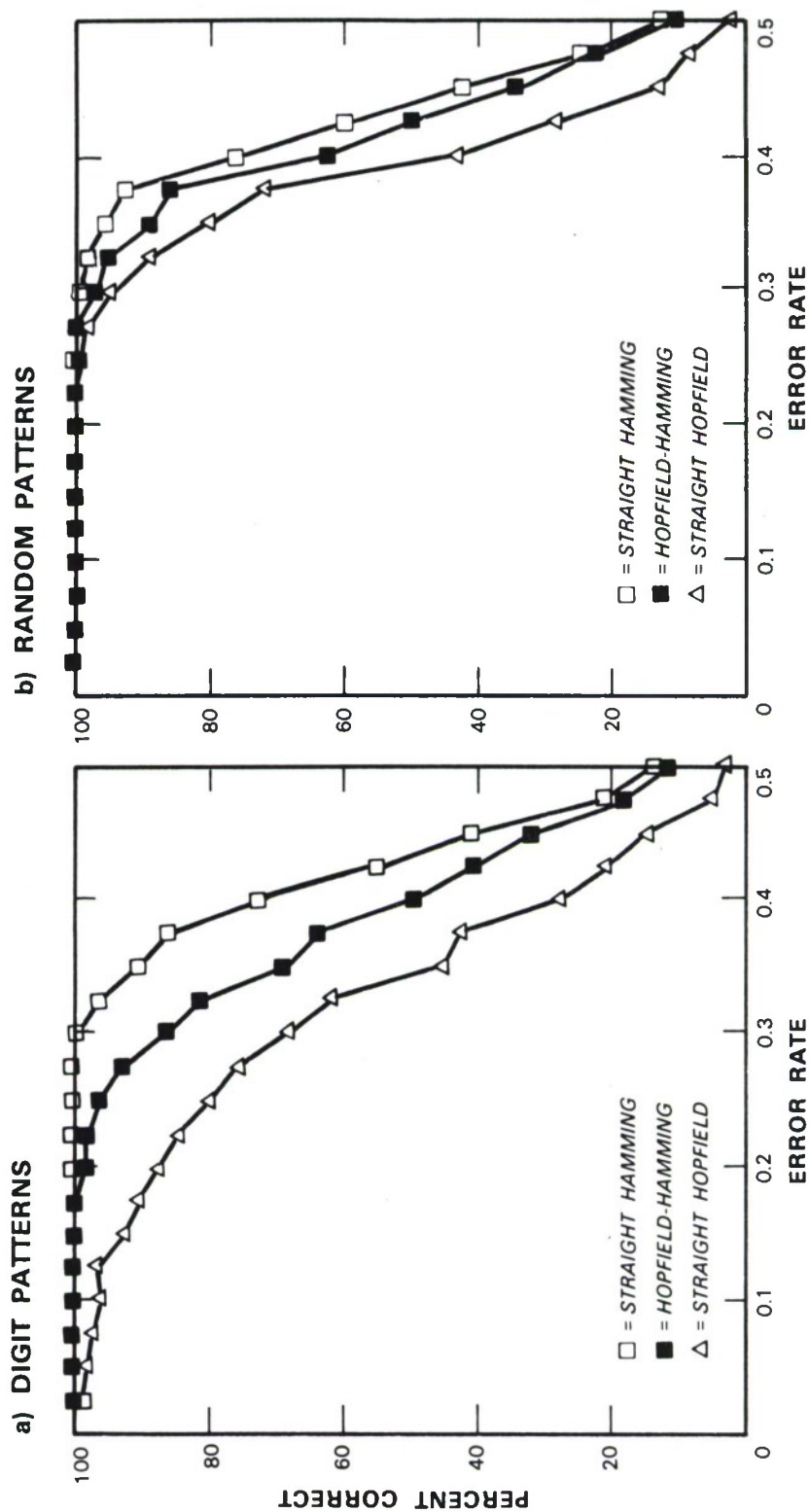


Figure 16. Results of Experiment No. 1 obtained using digit patterns (a) and random patterns (b) and a Hopfield net without orthogonalization.

the Hamming net is always correct until an error rate of .300. The results of the second experiment using random patterns are presented in Figure 16b. Here, the Hamming net begins failing at an error rate of about .300 as before, but the Hopfield-Hamming net survives until an error rate of .250 as opposed to .150, and the Hopfield net fails at .250 instead of .025, a marked improvement. Clearly, the Hopfield net is sensitive to a “well-behaved” set of Hamming distances among the stored states. In general, the straight Hamming net yields the best overall results.

Hopfield¹ suggested that the Hopfield net could be used as a bibliographic retrieval system. A third experiment was performed to test this conjecture. A network of 11 exemplars, each with 168 elements, was created from a bibliography based on the references in Hopfield’s paper¹. Two 16-character fields were used for the name (including two initials) and the publication, and a 2-character field was used for the date. The 16-character fields consisted of the letters “a” through “z” and the character “_” to mark the end of the character string if shorter than 16 characters. The 2-character field contained the last two digits of a 20th century date (0 through 9). Characters were represented by 5-bit bytes and digits by 4-bit bytes using ASCII-like representations. This resulted in a total of 168 bits. In order to maintain desirable Hamming distances, it was necessary to “pad out” a field with “unique” data when it was bigger than its entry. A simple reflection of the entry was performed to meet this requirement. The format is shown in Table V, which contains the 11 exemplars which were selected from a list of 27 references. References were selected only if they satisfied a range of Hamming distances. This range was initially set to a

TABLE V
Exemplar Bibliography Patterns Used In Experiment No. 3

jclonguethiggensprocroysoclondon68
g__willwacher__rehbiolocybernetics76
vbmountcastle__elthemindfulbrain__78
jaanderson__nosrepsychologyreview77
wbkristan__natsirinformationproce80
bwknight__thgink__lectmathlifescie75
idharmon__nomrah__neuraltheorymode64
wsmcculloch__hcolbulletinmathbiop43
m__minsky__yksnim__perceptrons__snor69
dohebb__bbeh__hebborganizationbeha49
psgoldman__namdlobrainresearch__hc77

minimum and had to be relaxed until 11 references were found to meet the criterion. The resulting Hamming distances covered the range from 71 through 93, a reasonable variation from the expected distance of 84 which random patterns would have produced. In all but one case, the entire information was exactly retrievable from the input of either the name with initials, the name without initials, or the publication. It was observed that a straight Hamming net would have sufficed.

Orthogonalization, described in the previous section, has two positive effects on a Hopfield net: (1) it improves the performance of an already stable network, and (2) it allows the number of exemplars to be increased without losing stability as long as $M \leq N$. Experiment No. 1 was repeated after orthogonalizing the net. This experiment involved the handcrafted digit patterns with a less than ideal set of Hamming distances. A dramatic improvement in the performance of the Hopfield net may be seen in Figure 17a. The Hopfield-Hamming net also improved in performance, as shown in Figure 17b.

Following the initial orthogonalization experiments we increased the number of exemplars to 16, dropping the square pattern and including all the hexadecimal digits (0 through F). These 16 exemplars remained stable. Performance statistics were gathered for error rates of .100 through .300; and, as we expected, the Hopfield and Hopfield-Hamming nets suffered a degradation in performance as illustrated in Figure 18a and Figure 18b, respectively.

The bibliography retrieval Hopfield net was also orthogonalized and expanded to include all 27 entries from Hopfield's paper. Hamming distances covered the range from 51 through 100, excluding three smaller distances due to the fact that three references had the same publication entry. All entries were retrievable from only the last name and initials. All entries but one were retrievable from only the last name. Four entries were not retrievable by publication alone, three of which shared the same publication and one of which differed from another publication by only three characters.

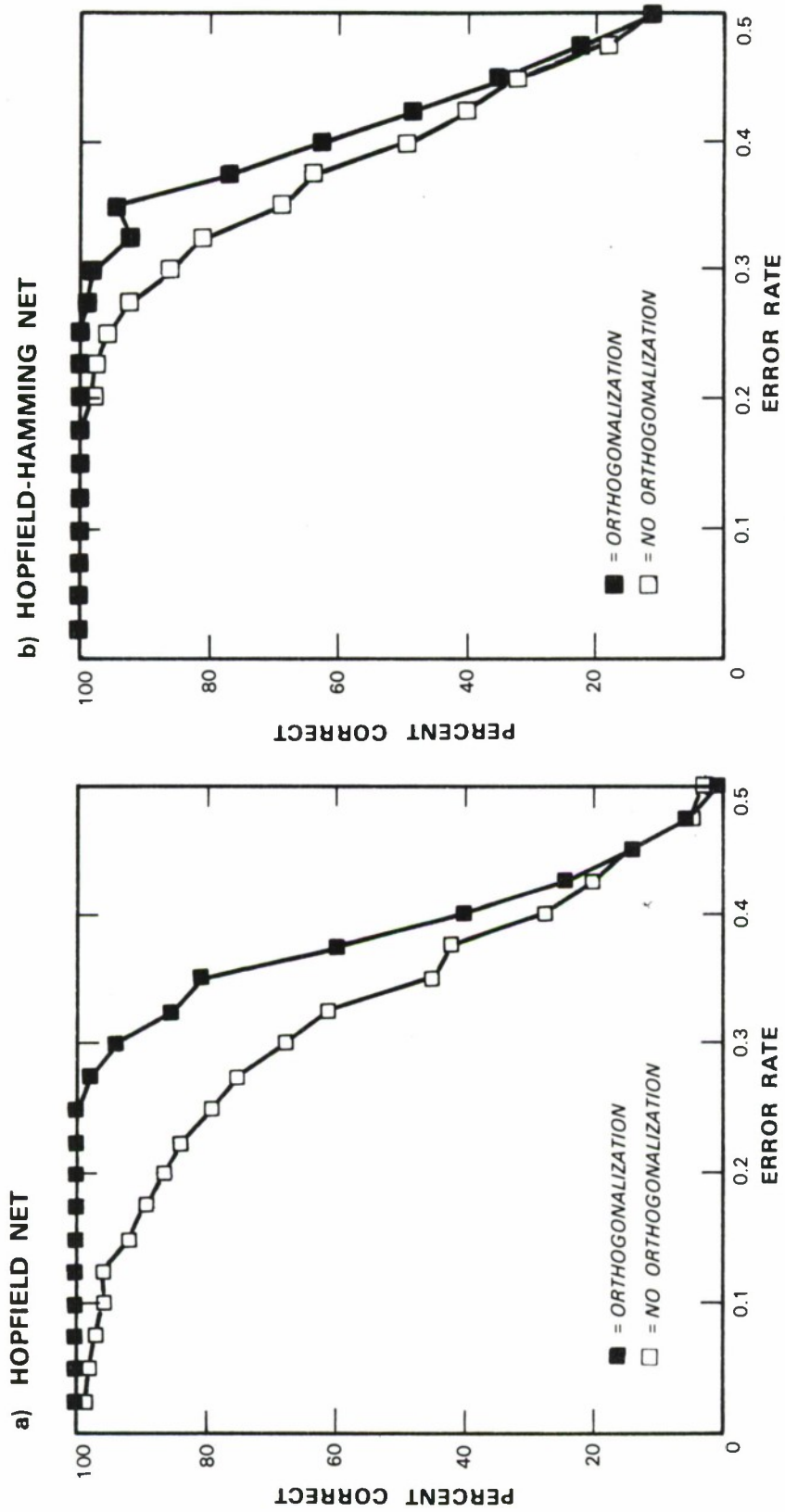


Figure 17. Hopfield net (a) and Hopfield-Hamming net (b) performance with and without orthogonalization using digit patterns from Experiment No. 1.

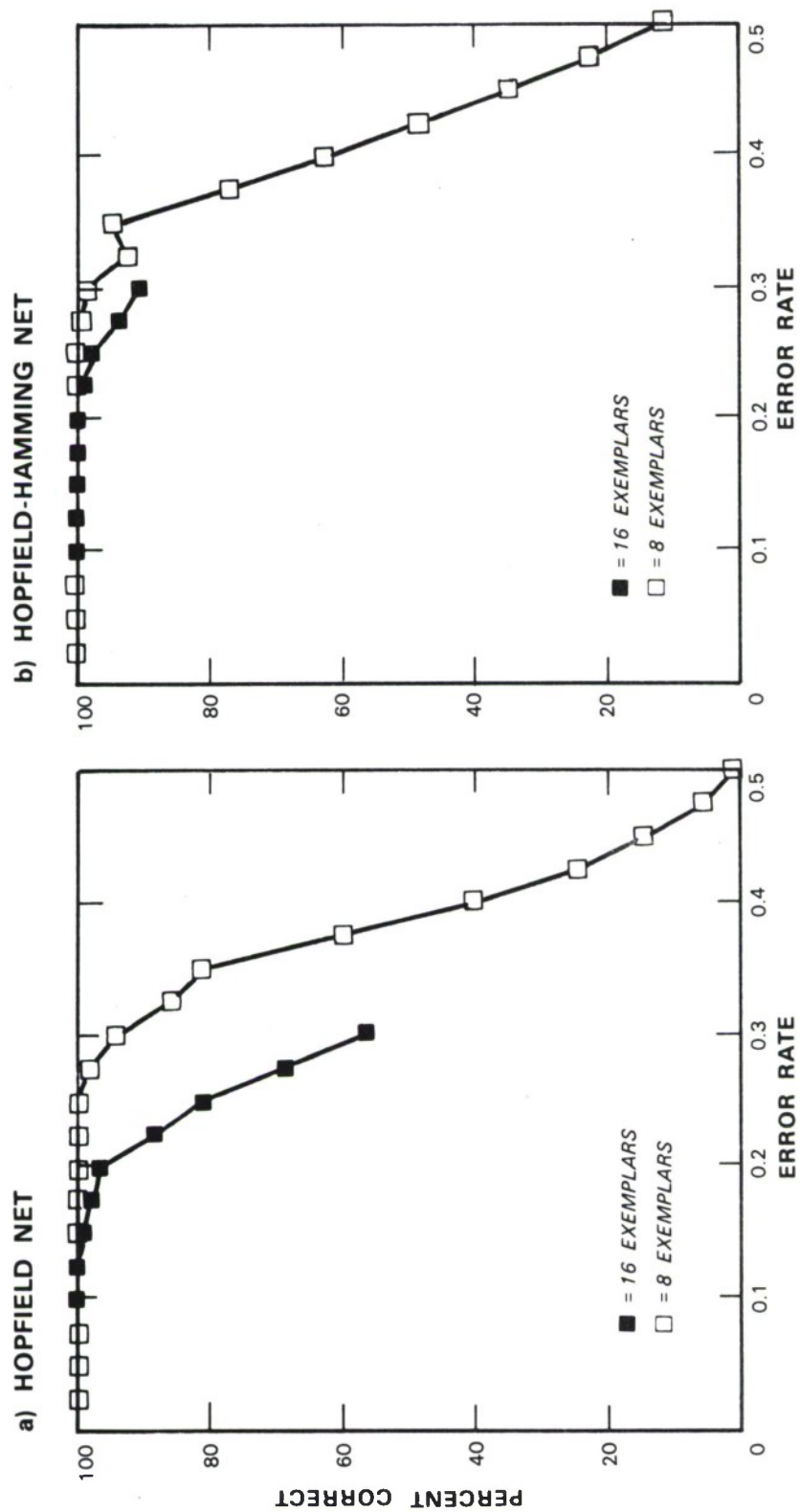


Figure 18. Hopfield net (a) and Hopfield-Hamming net (b) performance with orthogonalization using 16 hexadecimal digits and using 8 digit patterns as exemplars.

6. CONCLUSIONS

Two neural network approaches have been compared as they apply to several pattern classification problems. First, it was shown that for patterns consisting of statistically independent binary components, an optimum classifier can be configured as a single-layer perceptron followed by a densely-connected neuron-like net that labels the most likely stored patterns. We called this classifier a Hamming net and showed several methods of implementing this algorithm. We also compared implementation complexity of the Hamming net and a Hopfield net and concluded that the Hamming net was, in principle, a simpler device than a fully-connected Hopfield net. This was followed by a brief review of the Hopfield model and the discussion of several techniques for enhancing performance of this model. Finally, we presented experimental comparisons for three types of input data. For these data, it was seen that the storage prescription of¹ yielded poorer performance than did the Hamming net. By carefully choosing the stored states and by orthogonalization, Hopfield net performance became comparable to but never better than Hamming net performance for the examples chosen. Further research is needed to compare such systems when the neural components are analog rather than digital. Further research should also examine the performance of neural classification nets with analog, continuously variable, inputs.

REFERENCES

1. J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. National Acad. Sci. USA*, Vol. **79**, pp. 2554-2558 (April 1982).
2. J.J. Hopfield, D.I. Feinstein, and R.G. Palmer, "'Unlearning' has a Stabilizing Effect in Collective Memories," *Nature*, Vol. **304**, pp. 158-159 (July 1983).
3. J.J. Hopfield, "Neurons with Graded Response Have Collective Computational Abilities," *Proc. National Acad. Sci. USA*, Vol. **81**, pp. 3088-3092 (May 1984).
4. N. Farhat, S. Miyahara, and K.S. Lee, "Two-Dimensional Optical Implementation of Neural Networks and Their Application in Recognition of Radar Targets," in *Proc. of the Neural Networks for Computing Conf., Snowbird, UT* (April 1986).
5. B. Gold, "Hopfield Model Applied to Vowel and Consonant Discrimination," in *Proc. of the Neural Networks for Computing Conf., Snowbird, UT* (April 1986).
6. P.M. Grant and J.P. Sage, "A Comparison of Neural Network and Matched Filter Processing for Detecting Lines in Images," in *Proc. of the Neural Networks for Computing Conf., Snowbird, UT* (April 1986).
7. J.J. Hopfield and D.W. Tank, "Computing with Neural Circuits: A Model," *Science*, Vol. **233**, pp. 625-633 (August 1986).
8. D.W. Tank and J.J. Hopfield, "Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Trans. on Circuits and Systems*, Vol. **CAS-33**, pp. 533-541 (May 1986).
9. R.G. Gallager, *Information Theory and Reliable Communication*, John Wiley and Sons, New York (1968).
10. T. Martin, *Acoustic Recognition of a Limited Vocabulary in Continuous Speech*, Ph.D. Thesis, Dept. of Electrical Engineering, University of Pennsylvania (1970).
11. M. Minsky and S. Papert, *Perceptrons, An Introduction to Computational Geometry*, MIT Press, Cambridge, MA (1986).
12. F. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, New York (1959).
13. T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, NY (1984).

